.szer Znaczenie specyfikacji *szer* zależy od typu konwersji:
- maksymalna szerokość pola (`%s`);
- liczba cyfr po kropce dziesiętnej (`%e`, `%E`, `%f`);
- maksymalna liczba cyfr znaczących po kropce dziesiętnej (`%g`, `%G`);
- minimalna liczba cyfr (`%d`, `%o`, `%i`, `%u`, `%x`, `%X`), w razie potrzeby uzupełniana nieznaczącymi zerami

W wersji `gawk` *wspierana jest dynamiczna specyfikacja ANSI C dla długość i szer. Znak* `*` *zamiast długość lub szer spowoduje pobranie odpowiedniej wartości z listy argumentów.*

Funkcje
Definicja funkcji może być umieszczona w dowolnym miejscu programu. Funkcje są definiowane następująco:

`function` *nazwa* `(`*lista-argumentów*`) { `*lista-instrukcji*` }`

lista-argumentów to ciąg oddzielonych przecinkami argumentów funkcji. Podczas wywołania funkcji, argumentom nadawane są odpowiednie wartości. Nazwy argumentów są lokalne; są przekazywane przez wartość, z wyjątkiem tablic, które są przekazywane „przez referencję".

Lista-instrukcji może zawierać instrukcję `return` *wyrażenie*. Wykonanie `return` polega na obliczeniu wartości *wyrażenia* i przekazaniu tej wartości w miejsce wywołania funkcji. *Wyrażenie* jest opcjonalne – jeżeli go nie ma, `return` przekazuje tylko sterowanie do miejsca wywołania. Jeżeli nie ma `return` sterowanie jest przekazywane do miejsca wywołania, a wartość zwracana jest nieokreślona.

Funkcje mogą być użyte w dowolnym wyrażeniu w obu częściach par *wzorzec-procedura* a także wewnątrz innych funkcji. Dozwolona jest rekursja.

Zmienne lokalne są deklarowane jako „nadmiarowe" parametry formalne Wywołanie funkcji z mniejszą od deklarowanej liczbą parametrów jest poprawne – wszystkie nadmiarowe parametry przyjmują wartości zerowe. Zwyczajowo nadmiarowe parametry oddziela się dodatkowymi znakami odstępu:

```
# a, b, c są lokalne
function qq (x, y,   a, b, c) { ... }
/abc/{...; qq(1,2); ... }
```

Przy wywołaniu funkcji *nie można* umieszczać odstępu pomiędzy jej nazwą a rozpoczynającym listę argumentów nawiasem `(`. Wymóg ten, który *nie* dotyczy *funkcji wbudowanych* jest spowodowany uniknięciem dwuznaczności związanej z operatorem konkatenacji.

Zmienne środowiskowe
Zmienna `AWKPATH` *określa katalogi, w których będą szukane pliki podane za pomocą opcji* `-f`*. Domyślną wartością tej zmiennej jest* `".:/usr/local/share/awk"`*. Jeżeli nazwa podana jako argument opcji* `-f` *zawiera znak* `/`*, żadne katalogi nie są przeszukiwane.*

Zasoby internetowe
`ftp://ftp.gnu.ai.mit.edu/pub/gnu/gawk-3*.tar.gz`.
`ftp://ftp.whidbey.net/pub/brennan/mawk*.tar.gz`.

Opracowanie: Tomasz Przechlewski

Ściąga do AWK*

Uruchamianie programu
Jeżeli program AWK-owy jest krótki, to najprościej jest umieścić go pomiędzy znakami *pojedynczego* cudzysłowa w linii poleceń, jeżeli jest dłuższy wpisać do pliku i uruchamiać z opcją `-f`:

`awk '`*program*`' `*plik1 plik2 ...*
`awk -f `*plik-z-programem plik1 plik2 ...*

Argumenty wywołania programu
Standardowe argumenty wywołania programu umożliwiają określenie separatora pól, przypisanie zmiennym początkowych wartości oraz wskazanie pliku źródłowego z programem AWK-owym.

Argumenty wywołania programu

Argument	Znaczenie
`-F` *sp*	separatorem pól będzie *sp*.
`-v`*zm*`=`*war*	przypisuje zmiennej *zm* wartość *war*, przed rozpoczęciem wykonywania programu.
`-f` *plik*	czyta program źródłowy z *pliku*. Opcję `-f` można podać wielokrotnie.
`--`	koniec opcji.

Wykonanie programu
Programy w AWK składają się z par *wzorzec-procedura* oraz, opcjonalnie, z definicji funcji:

wzorzec`{`*procedura*`}`
`function` *nazwa*`(`*parametry-formalne*`){`*polecenia*`}`

Kolejność wykonania programu jest następująca. Najpierw przypisywane są wartości wszystkim zmiennym, określonym w opcji `-v`. Następnie wykonywane są procedury ze wzorców `BEGIN` (jeżeli takie znajdują się w programie). Z kolei czytane są pliki od 1 do `ARGC-1` z tablicy `ARGV`. Modyfikowanie `ARGC`/`ARGV` umożliwia kontrolę nad plikami wejściowymi, które mają być wczytane przez AWK.

Jeżeli argumenty wywołania programu mają postać *zm*`=`*wart*, to taki argument jest traktowany jako przypisanie zmiennej *zm* wartości *wart*. Przypisanie to ma miejsce po wykonaniu procedury `BEGIN`.

Jeżeli wartością elementu `ARGV` jest napis pusty, AWK pomija go i nie traktuje jako nazwy pliku do przeczytania.

Jeżeli program składa się wyłącznie ze wzorca (lub wzorców) `BEGIN`, to AWK nie przetwarza żadnych plików wejściowych. Jeżeli program składa się wyłącznie ze wzorców `END` to AWK czyta dane wejściowe (ze standardowego wejścia).

Wyrażenia regularne
Składnia wyrażeń regularnych jest poszerzoną wersją z programu `egrep`. Znak tworzący słowo to: litera, cyfra oraz znak podkreślenia.

Wyrażenia regularne wg. malejącej kolejności wykonywania

Wyrażenie	Znaczenie
`(`*r*`)`	*r* (nawiasy służą do grupowania wyrażeń)
`c`	znak nie będący metaznakiem
`\c`	wyłącza metaznaczenie znaku `c`
`^`	początek napisu
`$`	koniec napisu
`.`	dowolny znak
`[ab...]`	dowolny ze znaków `a`, `b`... Można stosować skrócony zapis `[a-z]` oznaczający znak z zakresu `a`–`z`

* Opracowana na podstawie „AWK reference card" Arnolda Robbinsa. W tekście przedstawiono standard AWK-a oraz rozszerzenia interpretera `gawk`. Opis rozszerzeń jest oznaczony *za pomocą pisma pochyłego.*

`[^ab...]`	dowolny znak oprócz `a`, `b`... Zapis skrócony: `[^a-z]`
`\y`	*granica słowa*
`\B`	*wnętrze słowa*
`\<`	*początek słowa*
`\>`	*koniec słowa*
`\w`	*znak alfanumeryczny*
`\W`	*wszystkie znaki za wyjątkiem alfanumerycznych*
`\‘`	*początek bufora (napisu)*
`\’`	*koniec bufora (napisu)*
r`*`	zero lub więcej napisów pasujących do *r*
r`+`	jeden lub więcej napisów pasujących do *r*
r`?`	zero lub jeden napis pasujący do *r*
`r{`*n*`,`*m*`}`	*od n do m powtórzeń r*
r1`|`*r2*	*r1* lub *r2* (*r* oznacza wyrażenie regularne)

Rekordy

Domyślnie rekordy są odzielone znakami nowej linii. Wartość zmiennej `RS` przechowuje napis używany do oddzielania poszczególnych rekordów. Jeżeli `RS` jest pojedynczym znakiem wtedy ten znak oddziela rekordy; w przeciwnym wypadku `RS` jest traktowany jak wyrażenie regularne. Jeżeli `RS=""` (napis pusty) wtedy, separatorami rekordów są puste linie (jedna lub więcej). Jeżeli `RS` jest napisem pustym, to znaki końca linii zawsze są traktowane jako separatory pól bez względu na to jaka jest wartość zmiennej `FS`.

Jeżeli `RS` jest wyrażeniem regularnym, to zmienna `RT` zawiera napis będący separatorem bieżącego rekordu od rekordu następnego.

Pola

Po przeczytaniu każdego rekordu AWK dzieli go na *pola* wykorzystując jako separatory wartość zmiennej `FS`. Jeżeli `FS` jest pojedynczym znakiem, to ten znak jest separatorem pól; w przeciwnym wypadku `FS` jest traktowany jak wyrażenie regularne. W specjalnym przypadku, gdy `FS=" "` (dokładnie jedna spacja) to separatorem pól jest dowolnej długości ciąg odstępów (tj. spacji, znaków tabulacji lub nowego wiersza). Wiodące i końcowe (*trailing*) odstępy są pomijane. Na sposób w jaki rekordy są dzielone na pola ma wpływ wartość zmiennej `IGNORECASE`, w przypadku gdy `FS` jest wyrażeniem regularnym.

Do każdego pola można się odwołać za pomocą zmiennych `$1` (pierwsze pole), `$2` (drugie pole), `$3` itd. Zmienna `$0` oznacza cały rekord. Polom mogą być także przypisane nowe wartości. Zmiennej `NF` jest przypisywana liczba pól w bieżącym rekordzie.

Odwołanie się do pól nie istniejących (o numerze większym od `NF`) powoduje utworzenie napisów pustych. Przypisanie wartości polom nie istniejącym powoduje także odpowiednie zwiększenie wartości zmiennej `NF`, ewentualne utworzenie pól „pośrednich” (tj. pól od numeru `NF` +1 do przedostatniego) o wartości równej napisowi pustemu oraz uaktualnienie zmiennej `$0`. Odwołanie się do pola o numerze ujemnym powoduje błąd. Zmniejszenie wartości `NF` powoduje utratę „nadmiarowych” pól.

Wzorce

Wzorce mogą mieć postać:

`BEGIN` `END` *wyrażenie* *wzorzec1*`,` *wzorzec2*

Wzorce `BEGIN` i `END` są wzorcami specjalnymi, wykonywanymi – odpowiednio – przed i po przeczytaniu strumienia danych wejściowych. Wzorce `BEGIN` i `END` *muszą* posiadać procedury. Oba wzorce mogą występować wielokrotnie; są wtedy łączone i wykonywane tak jakby tworzyły jedną dużą procedurę. Mogą występować w dowolnym miejscu programu, włączając w to różne pliki źródłowe.

Wyrażenie we wzorcu może być dowolnym wyrażeniem zgodnie z opisem w punkcie Wyrażenia.

Konstrukcję *wzorzec1*`,`*wzorzec2* nazywamy wzorcem z przecinkiem. Do tego wzorca pasują wszystkie rekordy pasujące od *wzorzec1* do rekordu pasującego do *wzorzec2*, łącznie z tymi rekordami. W roli

`printf` *format*`,` *lista-wyrażeń*
: drukuje *listę-wyrażeń* według specyfikacji zawartej w *formacie*; patrz Funkcja printf.

`system(`*polecenie*`)`
: wykonuje polecenie systemowe, zwraca status zakończenia.

Przekierowanie strumienia wejścia/wyjścia za pomocą poleceń `print` oraz `printf`:

`print >` *plik*
: drukowanie do pliku. Pierwszy zapis powoduje skasowanie poprzedniej zawartości pliku (jeżeli istniał) lub utworzenie nowego, kolejne powodują dopisywanie do pliku.

`print >>` *plik*
: dopisywanie do pliku.

`print |` *prog*
: drukowanie w potoku (*prog* oznacza zewnętrzny program-odbiorcę strumienia danych).

Funkcja printf

Ogólna postać funkcji `printf` jest następująca:

`printf (`*format*`,` *arg1*`,`*arg2*`,...)`

Nawiasy okrągłe są opcjonalne. Napis lub zmienna napisowa *format* określa sposób przekształcania i formatowania argumentów. Zawiera on znaki kopiowane przy drukowaniu oraz specyfikacje przekształceń, z których każda określa sposób wypisania kolejnego argumentu. Specyfikacja ta rozpoczyna się od znaku `%` a kończy znakiem określającym typ konwersji.

Znaki konwersji

Znak	Typ przekształcenia
`%c`	znak ASCII
`%d`, `%i`	liczba całkowita
`%e`	liczba postaci `[-]d.dddddde[+-]dd`
`%E`	jak `e` ale wykorzystuje `E` zamiast `e`
`%f`	liczba postaci `[-]ddd.dddddd`
`%g`	`e` lub `f`, w zależności od tego które jest krótsze przy czym nieznaczące zera nie są drukowane
`%G`	jak `%g` ale wykorzystuje `%E` zamiast `%e`
`%o`	liczba ósemkowa bez znaku
`%s`	napis (ciąg znaków)
`%x`	liczba szestnastkowa bez znaku
`%X`	jak `%x`, ale wykorzystuje `ABCDEF` do oznaczenia liczb z przedziału 10–15
`%%`	literalnie znak `%`

Dodatkowo, pomiędzy znakiem `%` a znakiem określającym typ konwersji mogą wystąpić następujące znaki modyfikujące:

Znak	Typ przekształcenia
`-`	zawartość pola jest justowana do lewego krańca pola
spacja	dla wartości numerycznych, wstawia spację przed wartościami dodatnimi oraz minus przed ujemnymi
`0`	zawartość pola zamiast spacjami jest wypełniana zerami. Dotyczy także pól nieliczbowych
długość	Określenie minimalnego rozmiaru pola w znakach. Argument będzie wpisany do pola o długości *co najmniej* równej *długość*. Jeżeli argument składa się z mniejszej liczby znaków, to będzie ono uzupełnione do długości minimalnej znakami spacji. Jeżeli specyfikacja długości pola rozpoczyna się cyfrą 0, to pole będzie wypełniane zerami;

zmienna przyjmuje iteracyjnie wszystkie wartości indeksów *tablicy*. Kolejność przeglądania tablicy nie jest ustalona i jest zależna od konkretnej implementacji AWK-a. Działanie pętli jest *nieokreślone* jeżeli wewnątrz pętli zostaną dodane kolejne elementy do *tablicy*.

Element tablicy możemy usunąć za pomocą instrukcji:
`delete` *tablica*`[`*indeks*`]`

Konwersja i porównanie

Zmienne i pola mogą zawierać liczby (zniennoprzecinkowe), napisy lub oba te typy danych. Interpretacja wartości zmiennej jest określona przez kontekst. Jeżeli zmienna jest częścią wyrażenia numerycznego to będzie traktowana jako liczba, natomiast wykorzystana w wyrażeniu napisowym – jako napis.

Dodanie do zmiennej `0` wymusza konwersję jej wartości do liczby; konkatenacja zmiennej z napisem pustym powoduje traktowanie zmiennej jako napisu.

Do konwersji napisu do liczby wykorzystywana jest funkcja *atof(3)*. Liczba do napisu jest konwertowana za pomocą funkcji *sprintf(3)* wykorzystującej wartość zmiennej `CONVFMT` jako format konwersji.

Porównanie jest wykonywane następująco: jeżeli obie porównywane zmienne są numeryczne, są porównywane numerycznie. Jeżeli jedna jest zmienną numeryczną a druga zawiera „napis numeryczny", wtedy także porównanie jest numeryczne. W pozostałych wypadkach, wartość numeryczna jest konwertowana do napisu, a następnie wykonywane jest porównanie napisów. Dwa napisy są porównywane jako napisy. Według standardu POSIX, dwa „napisy numeryczne" winny być porównywane numerycznie. Ani `gawk` ani `mawk` nie stosuje się do tego zalecenia.

Nie zainicjalizowane zmienne mają wartość numeryczną `0` lub napisową `""` (napis pusty).

Wejście

`close(`*plik*`)`
: zamyka wejściowy plik lub potok.

`getline`
: wczytuje następny rekord z bieżącego pliku do `$0`, nadaje wartości `NF`, `NR`, `FNR`.

`getline` *z*
: wczytuje następny rekord do zmiennej *z*, nadaje wartości `NR`, `FNR`.

`getline <` *plik*
: wczytuje następny rekord z *pliku*, nadaje wartości `$0`, `NF`.

`getline` *z* `<` *plik*
: wczytuje następny rekord z *pliku* do zmiennej *z*.

prog `| getline`
: wczytuje następny rekord z potoku generowanego przez *program* do `$0`, nadaje wartości `$0`, `NF`.

prog `| getline` *z*
: wczytuje następny rekord z potoku generowanego przez *program* do zmiennej *z*.

`getline` zwraca `0` na końcu pliku/strumienia danych lub `-1` w przypadku błędu (np. błąd otwarcia pliku).

Wyjście

`close(`*plik*`)`
: zamyka wyjściowy plik lub potok.

`print`
: drukuje bieżący rekord. Na końcu rekordu wstawiana jest wartość zmiennej `ORS` (domyślnie `\n`).

`print` *lista-wyrażeń*
: drukuje *listę-wyrażeń* (poszczególne wyrażenia na liście są oddzielone przecinkami). Pomiędzy wyrażenia wstawiana jest wartość zmiennej `OFS`, na końcu listy wartość zmiennej `ORS`.

wzorzec1/*wzorzec2* mogą wystąpić tylko wyrażenia: wzorce `BEGIN` i `END` oraz wzorzec z przecinkiem są niedozwolone.

Symbole sterujące

Wewnątrz stałych napisowych (`"..."`) oraz „literałów regularnych" (`/.../`) można wykorzystać symbole sterujące do wstawienia znaków specjalnych.

Symbole sterujące

Symbol	Znaczenie	Symbol	Znaczenie
`\a`	dzwonek	`\r`	powrót karetki
`\b`	znak cofnięcia	`\t`	tabulacja
`\f`	koniec strony	`\v`	tabulator pionowy
`\n`	koniec wiersza	`\\`	w tył-ciach (`\`)
`\`*ddd*	oktalnie *ddd*	`\x`*hh*	heksadecymalnie *hh*
`\"`	literalnie `"`	`\/`	ciach (`/`)

Wyrażenia

Wyrażenia arytmetyczne mogą być wykorzystane jako wzorce oraz jako warunki poleceń sterujących.

Wyrażenia

`()`	grupowanie
`$`	odwołanie do pola
`++ --`	inkrementacja, dekrementacja
`^ **`	potęgowanie
`+ - !`	jednoargumentowe operatory `+` `-`, logiczna negacja
`* / %`	mnożenie, dzielenie i modulo
`+ -`	dodawanie i odejmowanie
odstęp	łączenie napisów
`< >`	mniejszy, większy
`<= >=`	mniejszy-równy, większy-równy
`!= ==`	różny, równy
`~ !~`	operator pasowania, operator niepasowania
`in`	należenie do tablicy
`&&`	iloczyn logiczny
`||`	suma logiczna
`?:`	wyrażenie warunkowe
`= += -= *= /= %= ^= **=`	przypisanie

Funkcje wbudowane

Funkcje arytmetyczne

Funkcja	Wartość funkcji
`atan2(y,x)`	arcus tangens z y=x w zakresia $-\beta$ do β
`cos(x)`	cosinus z x, x w radianach
`sin(x)`	sinus z x, x w radianach
`exp(x)`	eksponent, czyli funkcja wykładnicza e^x
`int(x)`	część całkowita z x
`log(x)`	logarytm z x przy podstawie e
`sqrt(x)`	pierwiastek kwadratowy z x
`rand()`	przypadkowa liczba z przedziału $\langle 0; 1)$
`srand(x)`	wartość początkowa generatora liczb pseudolosowych

Funkcje napisowe

`gsub(`*r*`,`*s*`,`*t*`)`
: Zamienia wyrażenie regularne *r* na napis *s* w napisie *t*. Zwraca liczbę zamian. Wywołana z dwoma parametrami zamienia w `$0`.

`gensub(`*r*`,`*s*`,`*a*`,`*t*`)`
: *Uogólniona funkcja* `gsub`. *Zwraca zmieniony napis (nie modyfikuje oryginalnego napisu t!). Zamienia wyrażenie regularne r w oparciu*

o napis s, w napisie t (jeżeli nie ma t zamienia w `$0`*). Argument a określa, który z kolei podnapis pasujący do wyrażenia r ma być wymieniony. Jeżeli a jest napisem rozpoczynającym się od* `"g"` *(lub* `"G"`*) to wymieniane są wszystkie napisy pasujące do r.*

Funkcja `gensub` *umożliwia wstawienie fragmentów wyrażenia regularnego w napisie s. Jeżeli wyrażenie r podzielimy za pomocą nawiasów,* `( i )` *na części składowe, to te składowe mogą później pojawić się w napisie s. Oznaczamy je jako* `\n`*, gdzie n jest cyfrą od 1 do 9. Znaczenie tego jest takie, że napis pasujący do n-tej składowej jest kopiowany, z tekstu t do tekstu zwracanego przez funkcję. W efekcie możliwe są wszelkiego rodzaju zmiany kontekstowe. Symbol* `\0` *oznacza całe wyrażenie regularne r.*

`index(`*s*`,`*t*`)`
Zwraca numer pierwszego znaku napisu *t* w napisie *s*. Jeżeli *s* nie zawiera *t* zwraca `0`.

`length(`*s*`)`
Podaje długość napisu *s* lub długość `$0` (jeżeli nie podano `s`).

`match(`*s*`,`*r*`)`
Jeżeli *s* zawiera napis pasujący do *r*, to zwraca numer pierwszego znaku tego napisu; w przeciwnym razie zwraca `0`. Przypisuje wartości zmiennym `RSTART` i `RLENGTH`. `RSTART` jest równe wartości zwracanej przez funkcję, `RLENGTH` jest równe długości napisu pasującego do *r*.

`split(`*s*`,`*a*`,`*fs*`)`
Z napisu *s* tworzy tablicę napisów *a*, wykorzystując do separacji wyrażenie regularne *r*. Jeżeli `split` jest wywołane tylko z dwoma parametrami to separatorem jest wartość zmiennej `FS`.

`sprintf(`*format*`,` *lista-wyrażeń*`)`
Zwraca napis, sformatowany według specyfikacji *format*.

`sub(`*r*`,`*s*`,`*t*`)`
Jak `sub`, ale zamienia tylko pierwszy napis pasujący do *r* na *s*.

`substr(`*s*`,`*p*`,`*n*`)`
Zwraca napis wycięty z *s* począwszy od pozycji *p* o długości *n* znaków (lub do końca *s*, jeżeli argument *n* jest pominięty).

`tolower(`*s*`)`
Zwraca napis, w którym duże litery zostały zamienione na małe.

`toupper(`*s*`)`
Zwraca napis, w którym małe litery zostały zamienione na duże.

Instrukcje sterujące

`{` *instrukcje* `}`
grupowanie instrukcji.

`if (`*w*`)` *instrukcja*
jeśli wyrażenie *w* jest prawdziwe, wykonaj *instrukcję*.

`if (`*w*`)` *instrukcja1* `else` *instrukcja2*
wykonaj *instrukcję1* jeśli wyrażenie *w* jest prawdziwe, w wypadku przeciwnym *instrukcję2*.

`while (`*w*`)` *instrukcja*
jeśli wyrażenie *w* jest prawdziwe, wykonaj *instrukcję* i powtórz.

`for (`*w1*`;` *w2*`;` *w3*`)` *instrukcja*
równoważne instrukcji: *w1*`;` `while (`*w2*`) {`*instrukcja*`;` *w3*`}`.

`for (`*zmienna* `in` *tablica*`)` *instrukcja*
instrukcja jest wykonywana dla *zmiennej* przyjmującej kolejno wartości indeksów z *tablicy*.

`do` *instrukcja* `while (`*w*`)`
wykonaj *instrukcję* jeśli wyrażenie *w* jest prawdziwe i powtórz.

`break`
natychmiastowe wyjście z pętli `while`, `for`, `do`.

`continue`
kontynuacja iteracji w pętlach `while`, `for`, `do`.

`next`
rozpoczęcie następnej iteracji głównej pętli wejściowej.

`exit` *wyrażenie*
sterowanie jest przekazywane bezpośrednio do akcji `END`. Jeśli polecenia `exit` użyto w akcji `END`, program kończy działanie. Opcjonalne *wyrażenie* zwracane jest jako status programu.

`nextfile`
zakończenie czytania bieżącego pliku i przejście do następnego z podanych w linii poleceń, lub (dla ostatniego) przejście do wzorca `END`. *Wykonanie* `nextfile` *zmienia wartość zmiennej* `FILENAME`*, przypisuje* `FNR` *wartość 1 oraz zwiększa o 1 wartość* `ARGIND`.

Zmienne wbudowane

Zmienna	Opis znaczenia
`ARGC`	liczba argumentów wywołania programu.
`ARGV`	tablica argumentów wywołania programu, indeksowana od `0` do `ARGC` -1.
`ARGIND`	*index w* `ARGV` *odpowiadający bieżącemu plikowi.*
`CONVFMT`	format konwersji napisów do liczb, domyślnie `%.6g`.
`ENVIRON`	*tablica zmiennych środowiskowych.*
`ERRNO`	*napis z systemowym opisem błędu, w przypadku błędu wykonania polecenia* `getline` *lub* `close`.
`FIELDWIDTHS`	*rozdzielona spacjami lista długości pól. Przydatne w przypadku pól o stałych długościach.*
`FILENAME`	nazwa bieżącego pliku wejściowego. We wzorcu `BEGIN` zmienna `FILENAME` nie jest zdefiniowana.
`FNR`	numer bieżącego rekordu w bieżącym pliku.
`FS`	separator pól, zobacz Pola.
`IGNORECASE`	*jeżeli niezerowa, to wszystkie wyrażenia regularne i funkcje napisowe nie rozróżniają wysokości liter .*
`NF`	liczba pól w bieżącym rekordzie.
`NR`	liczba przeczytanych rekordów.
`OFMT`	specyfikacja formatu wydruku dla argumentów numerycznych polecenia `print` (domyślnie `%.6g`).
`OFS`	separator pól na wyjściu; domyślnie spacja.
`ORS`	separator rekordów na wyjściu; domyślnie znak nowej linii.
`RLENGTH`	por. opis funkcji `match` w punkcie Funkcje.
`RS`	separator rekordów; domyślnie znak nowej linii.
`RT`	*napis pasujący do wyrażenia* `RS`.
`RSTART`	por. opis funkcji `match` w punkcie Funkcje.
`SUBSEP`	separator indeksów tablic, domyślnie znak o kodzie `\034`, por. Tablice asocjacyjne.

Tablice asocjacyjne

Indeks tablicy to wyrażenie pomiędzy parą nawiasów kwadratowych. Jeżeli wyrażeniem jest lista, to subskrypt jest napisem powstałym z połączenia wartości składowych oddzielonych wartością zmiennej `SUBSEP`. W ten sposób symulowane są tablice wielowymiarowe. Przykładowo:

```
i="A"; j="B"; k="C"
x[i, j, k] = "A qq!\n"
```

przypisuje napis `"A qq!\n"` elementowi tablicy `x`, któremu odpowiada indeks `"A\034B\034C"`.

Wyrażenie *indeks* `in` *tablica* pozwala ustalić czy określony *indeks* występuje w *tablicy*. Jeżeli występuje to wartością wyrażenia jest 1, w wypadku przeciwnym 0.

`if (`*indeks* `in` *tablica*`) print` *tablica*`[`*indeks*`]`

w przypadku tablic wielowymiarowych, należy używać konstrukcji `(i,j) in tablica`.

Do iteracyjnego dostępu do wszystkich elementów tablicy służy specjalna forma pętli `for`:

`for (`*zmienna* `in` *tablica*`)` *instrukcja*